How to adapt your existing XSLT stylesheet to the specificities of XMLmind XSL-FO Converter

Hussein Shafie

XMLmind Software

July 23, 2024

Table of Contents

General recommendations

All XSL-FO processors (e.g. Apache FOP, RenderX XEP, Antenna House XSL Formatter or our own XMLmind XSL-FO Converter) have their limitations and their specificities. For example, Apache FOP does not support `table-layout="auto"` in `fo:tables`. This implies that you'll always have to adapt your existing XSLT stylesheet when you switch to a new XSL-FO processor.

If you want your XSLT stylesheet to work fine whatever the XSL-FO processor being used, please stick to the following recommendations:

1. Pass a `$foProcessor` parameter (value: `FOP`, `XEP`, `AHF` or our own `XFC`)[1] to your XSLT stylesheet in order to implement code which depends on the XSL-FO processor being used. Example:

```
<fo:table xsl:use-attribute-sets="tgroup">
  <xsl:call-template name="commonAttributes"/>
  ...
  <xsl:when test="$foProcessor eq 'FOP'">
    <xsl:attribute name="table-layout">fixed</xsl:attribute>
  ...
```

2. Keep the XSL-FO you generate with your XSLT stylesheet as simple as possible.

3. Stick to XSL-FO version 1.0 as very few XSL-FO processors fully support XSL-FO version 1.1[2].

4. When you find something which seems wrong with an XSL-FO processor, refer to its *conformance statement* because it is generally a documented limitation. Ours is found in "*XMLmind XSL-FO Converter Conformance Statement*".

5. When really needed to, use an extension to XSL-FO. Our extensions are introduced in the following sections of this document and are extensively documented in "*Implementation specificities*".

6. Test the output of your XSLT stylesheet with different XSL-FO processors (e.g. XMLmind XSL-FO Converter *and* Apache FOP). This helps finding validity problems in the XSL-FO you generate.

[1] Or use one XSLT stylesheet parameter per XSL-FO processor: `fop1-extensions=1`, `xep-extensions=1`, `axf-extensions=1`, etc, as done in the DocBook XSL Stylesheets.

[2] XMLmind XSL-FO Converter does not support XSL-FO version 1.1 at all.

Make `fo:list-blocks` behave like proper lists

When your XSLT stylesheet generates a `fo:list-block`, XMLmind XSL-FO Converter (**XFC** for short) translates it to a sequence of styled paragraphs looking like specified list items. However, you'll generally want these list items to behave as expected when edited in the word processor. For example, if a list item starts with a bullet, pressing **ENTER** at the end of this list item creates a new list item also starting with a bullet.

For **XFC**, achieving to generate styled paragraphs having a list item behavior is far from being obvious as a `fo:list-block/fo:list-item/fo:list-item-label` may contain any text you want. For example, a `fo:list-item-label` may contain: "`~1~`", "`~2~`", "`~3~`", etc, or it may contain "`5-A-a)`", "`5-A-b)`", "`5-A-c)`", etc.

By default, **XFC** tries to infer the type of the list by examining its first `fo:list-item-label`. However, this will work only if this first `fo:list-item-label` contains the most common bullets ("`-`", "`*`", "`•`", etc) or the simplest numberings ("`1`", "`1.`", "`1)`", "`A`", "`A.`", "`A)`", etc). For example, **XFC** heuristics fail for "`~1~`" and "`5-A-a)`".

Therefore it is recommended no to rely on **XFC** heuristics and instead, explicitly specify the *label format* used for all the items of a `fo:list-block`. This is done using `fo:list-block` extension attribute `xfc:label-format`[3]. Examples:

`xfc:label-format="~%{decimal}~"`
: Corresponding labels are: "`~1~`", "`~2~`", "`~3~`", etc.

`xfc:label-format="%{decimal;start=5}-"`
: Corresponding labels are: `5-`", "`6-`", "`7-`", etc.

`xfc:label-format="%{upper-alpha;inherit}-"`
: Corresponding labels for nested `fo:list-block` are: `5-A-`", "`5-B-`", "`5-C-`", etc.

`xfc:label-format="%{lower-alpha;inherit})"`
: Corresponding labels for nested `fo:list-block` are: `5-A-a)`", "`5-A-b)`", "`5-A-c)`", etc.

Note

Specified extension attribute `xfc:label-format` supersedes whatever text is found in the `fo:list-item-label` elements of a `fo:list-block` .

Actual XSLT code making use of extension attribute `xfc:label-format`: excerpts from the DocBook XSL Stylesheets as modified by XMLmind Software:

```
<fo:list-block id="{$id}" xsl:use-attribute-sets="orderedlist.properties">
  <xsl:attribute name="provisional-distance-between-starts">
    <xsl:value-of select="$label-width"/>
  </xsl:attribute>

  <xsl:call-template name="xfcLabelFormat">
```

[3] The "`xfc`" prefix is bound to the "`http://www.xmlmind.com/foconverter/xsl/extensions`" namespace.

```
    <xsl:with-param name="userLabelWidth" select="$label-width"/>
  </xsl:call-template>
  ...
```

where XSLT named template "`xfcLabelFormat`" is defined as:

```
<xsl:template name="xfcLabelFormat">
  <xsl:param name="userLabelWidth" select="$orderedlist.label.width"/>

  <xsl:if test="$xfc.extensions = 1 or
                contains($xfc.extensions, 'label-format')">
      <xsl:variable name="labelFormat">
        <xsl:variable name="num">
          <xsl:choose>
            <xsl:when test="@numeration = 'arabic'">decimal</xsl:when>
            <xsl:when test="@numeration = 'loweralpha'">lower-alpha</xsl:when>
            <xsl:when test="@numeration = 'lowerroman'">lower-roman</xsl:when>
            <xsl:when test="@numeration = 'upperalpha'">upper-alpha</xsl:when>
            <xsl:when test="@numeration = 'upperroman'">upper-roman</xsl:when>
            <xsl:otherwise>decimal</xsl:otherwise>
          </xsl:choose>
        </xsl:variable>

        <xsl:variable name="start">
          <xsl:if test="@continuation = 'continues'">
            <xsl:variable name="startNum">
              <xsl:call-template name="orderedlist-starting-number"/>
            </xsl:variable>

            <xsl:if test="$startNum &gt; 1">
              <xsl:value-of select="concat(';start=', $startNum)"/>
            </xsl:if>
          </xsl:if>
        </xsl:variable>

        <xsl:variable name="inherit">
          <xsl:if test="@inheritnum = 'inherit'">
            <xsl:value-of select="';inherit'"/>
          </xsl:if>
        </xsl:variable>

        <xsl:value-of select="concat('%{', $num, $start, $inherit, '}.')"/>
      </xsl:variable>

      <xsl:attribute name="xfc:label-format"
        xmlns:xfc="http://www.xmlmind.com/foconverter/xsl/extensions">
        <xsl:value-of select="$labelFormat"/>
```

```
42        </xsl:attribute>
43    </xsl:if>
44    ...
45  </xsl:template>
```

More information about extension attribute `xfc:label-format`.

Workaround the problem of `fo:block` divisions having background colors and/or borders

An XSLT stylesheet often translates a note, admonition or sidebar, that is, a *division* possibly containing paragraphs, lists and tables, to a `fo:block` possibly containing `fo:blocks`, `fo:list-blocks` and `fo:tables`. The problem is that this easy and straightforward approach is not compatible with the document model of most word processors.

For a word processor like MS-Word, a document is a *flat list*[4] of styled paragraphs and styled tables. Of course it's the job of XMLmind XSL-FO Converter (**XFC** for short) to cope with the specificities of its output formats. However when the `fo:block` division is given a background color and/or borders (which is often the case), this gives very poor results.

A workaround for this **XFC** limitation is to modify the XSLT stylesheet to make it translate a note, admonition or sidebar to a `fo:table` containing a single `fo:table-cell`[5]. This workaround works well but is tedious to implement “by hand”.

Enters `fo:block` extension attribute `xfc:render-as-table`. This boolean extension attribute simply instructs **XFC** to process a `fo:block` having attribute `xfc:render-as-table="true"` to as if it were a `fo:table` containing a single `fo:table-cell`. In other words, it makes the above workaround very easy to implement.

Actual XSLT code making use of extension attribute `xfc:render-as-table`: excerpts from the XSLT 2.0 stylesheets of XMLmind DITA Converter:

```
<fo:block xsl:use-attribute-sets="note note-with-borders">
  <xsl:if test="$foProcessor eq 'XFC'">
    <xsl:attribute name="xfc:render-as-table"
                   select="if ($xfcRenderAsTable = 'note')
                           then 'true'
                           else 'false'"/>
  </xsl:if>

  <xsl:call-template name="commonAttributes"/>

  <fo:block xsl:use-attribute-sets="note-head">
    <xsl:value-of select="$label" />
  </fo:block>
  ...
```

More information about extension attribute `xfc:render-as-table`.

[4] A paragraph may *not* contain sub-paragraphs and tables.

[5] A table cell may, of course, contain paragraphs and sub-tables.

Workaround the problem of `fo:leaders`

XSLT stylesheets often automatically generate a table of content, a list of figures, a back of the book index, etc. These lists always reference page numbers and these page numbers are generally aligned to the right by the means of `fo:leader` elements. Example:

```
<fo:block ...>
  <fo:inline>Part I. </fo:inline>
  <fo:basic-link internal-destination="part1">Part 1</fo:basic-link>
  <fo:leader leader-pattern="dots" keep-with-next.within-line="always"/>
  <fo:page-number-citation ref-id="part1"/>
</fo:block>

<fo:block ...>
  <fo:inline>Chapter 1. </fo:inline>
  <fo:basic-link internal-destination="chapter1">Chapter 1</fo:basic-link>
  <fo:leader leader-pattern="dots" keep-with-next.within-line="always"/>
  <fo:page-number-citation ref-id="chapter1"/>
</fo:block>
```

The RTF, WML, DOCX and ODT file formats do not support leader objects. That's why XMLmind XSL-FO Converter (**XFC** for short) implements `fo:leaders` by means of tab stops. *This implies that you'll have to help **XFC** position these tab stops*. This is done using `fo:leader` extension attributes `xfc:tab-position` and `xfc:tab-align`.

Actual XSLT code making use of these extension attributes: excerpts from the XSLT 2.0 stylesheets of XMLmind DITA Converter:

```
<xsl:template match="ditac:tocEntry" mode="fmbmTOC">
...
          <fo:leader leader-pattern="dots"
                     keep-with-next.within-line="always">
            <xsl:if test="$foProcessor eq 'XFC'">
              <xsl:attribute name="xfc:tab-position">-30pt</xsl:attribute>
              <xsl:attribute name="xfc:tab-align">right</xsl:attribute>
            </xsl:if>
          </fo:leader>
          <xsl:text> </xsl:text>
          <fo:page-number-citation ref-id="{$id}"/>
```

- `xfc:tab-position="-30pt"` specifies that the tab position is `30pt` before to the right margin. (A positive value specifies the tab position relative to the left margin.)
- `xfc:tab-align="right"` specifies that the content following the tab, that is, `fo:page-number-citation`, is right-aligned. (By default, the content following a tab is left aligned.)

More information about extension attributes `xfc:tab-position` and `xfc:tab-align`.

Mark your headings

The RTF, WML, DOCX or ODT document you'll generate using XMLmind XSL-FO Converter will be nicer to edit and navigate for the user of the word processor if you mark chapter and section titles as being *headings*. This is done using `fo:block` extension attribute `xfc:outline-level`.

The value of this attribute is an integer between 1 and 9. A top-level heading, for example, the title of the chapter of a book must be given `xfc:outline-level="1"`. The title of a section directly contained in a chapter must be given `xfc:outline-level="2"`. The title of a sub-section must be given `xfc:outline-level="3"` and so forth.

Actual XSLT code making use of extension attribute `xfc:outline-level`: excerpts from XMLmind's XSLT 2.0 stylesheets converting XHTML to XSL-FO:

```
<xsl:template match="html:h1">
  <fo:block xsl:use-attribute-sets="h1">
    ...
    <xsl:call-template name="commonAttributes"/>
    <xsl:call-template name="xfcOutlineLevel"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

where XSLT template "`xfcOutlineLevel`" is defined as:

```
<xsl:template name="xfcOutlineLevel">
  <xsl:if test="$foProcessor eq 'XFC' and $set-outline-level eq 'yes'">
    <xsl:variable name="sectioningRoot"
      select="(ancestor::*[self::html:body or
                           self::html:blockquote or
                           self::html:details or
                           self::html:dialog or
                           self::html:fieldset or
                           self::html:figure or
                           self::html:td])[last()]"/>

    <!-- Ignore headings which are contained in sectioning roots
         other than the body (e.g. a legend). -->
    <xsl:variable name="isHeading"
      select="exists($sectioningRoot/self::html:body) or
              ($root-id ne '' and
               exists($sectioningRoot/descendant-or-self::*[@id eq $root-id]))"/>

    <xsl:if test="$isHeading">
      <!-- This a gross simplification as the type of heading being used
           (h1, h2, ..., h6) does not always specify the outline level. -->
      <xsl:attribute name="xfc:outline-level"
                     select="substring-after(local-name(), 'h')"/>
```

```
    </xsl:if>
  </xsl:if>
</xsl:template>
```

More information about extension attribute `xfc:outline-level`.

Do not forget about the spell checker

Do not forget that word processors always have spell checkers. Therefore if you don't want the user see zillions of errors after opening in a word processor a document created using XMLmind XSL-FO Converter, please make sure to follow these recommendations:

1. Attribute `xml:lang` (or equivalently attribute `language` AND attribute `country`) *must be specified **at least** on the `fo:root` element.*
2. Your XSLT stylesheet must process `xml:lang` attributes or `lang` attributes found on the source XML elements by adding equivalent `xml:lang` attributes to the target XSL-FO elements.

Excerpts from the XSLT 2.0 stylesheets of XMLmind Ebook Compiler:

```
<xsl:template match="html:html">
  <fo:root>
    <xsl:call-template name="localizationAttributes"/>
  ...
```

and (for *all* the templates matching an XHTML element; below is just the template matching HTML `span`):

```
<xsl:template match="html:span">
  <fo:inline xsl:use-attribute-sets="span">
    ...
    <xsl:call-template name="commonAttributes"/>
    <xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```

where XSLT templates "`localizationAttributes`" and "`commonAttributes`" are defined as:

```
<xsl:template name="commonAttributes">
  <xsl:call-template name="idAttribute"/>
  <xsl:call-template name="localizationAttributes"/>
  <xsl:call-template name="xmlSpaceAttribute"/>
</xsl:template>

<xsl:template name="localizationAttributes">
  <xsl:choose>
    <xsl:when test="exists(@lang)">
      <xsl:attribute name="xml:lang" select="string(@lang)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy-of select="@xml:lang"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

More information in "*Adding language information to the files created by **XFC***".